Error Handling, Auditing and Logging

Objective

Many industries are required by legal and regulatory requirements to be:

* Auditable – all activities that affect user state or balances are formally tracked
* Traceable – it’s possible to determine where an activity occurs in all tiers of the application
* High integrity – logs cannot be overwritten or tampered by local or remote users

Well-written applications will dual-purpose logs and activity traces for audit and monitoring, and make it easy to track a transaction without excessive effort or access to the system. They should possess the ability to easily track or identify potential fraud or anomalies end-to-end.

Environments Affected

All.

Relevant COBIT Topics

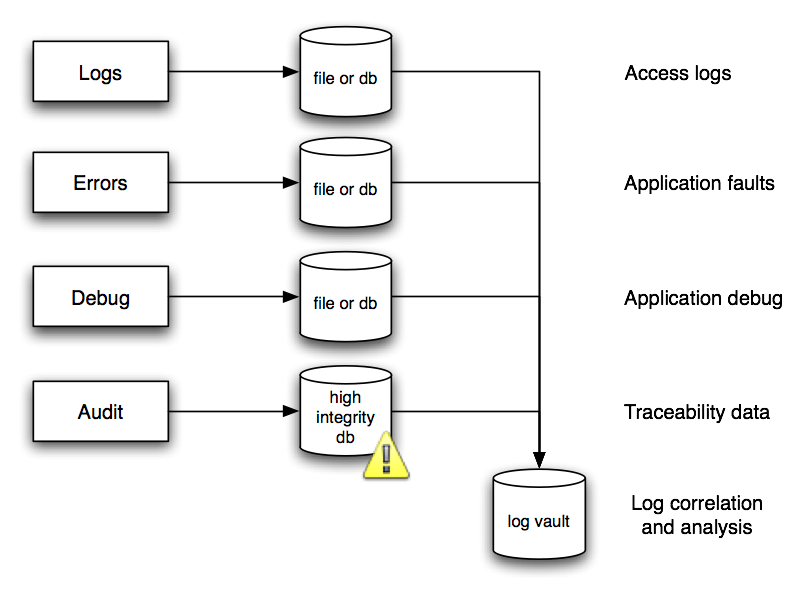
DS11 – Manage Data – All sections should be reviewed, but in particular:

DS11.4 Source data error handling

DS11.8 Data input error handling

Description

Error handling, debug messages, auditing and logging are different aspects of the same topic: how to track events within an application:



Best practices

* Fail safe – do not fail open
* Dual purpose logs
* Audit logs are legally protected – protect them
* Reports and search logs using a read-only copy or complete replica

Error Handling

Error handling takes two forms: structured exception handling and functional error checking. Structured exception handling is always preferred as it is easier to cover 100% of code. Functional languages such as PHP 4 that does not have exceptions are very hard to cover 100% of all errors. Code that covers 100% of errors is extraordinarily verbose and difficult to read, and can contain subtle bugs and errors in the error handling code itself.

Motivated attackers like to see error messages as they might leak information that leads to further attacks, or may leak privacy related information. Web application error handling is rarely robust enough to survive a penetration test.

Applications should always fail safe. If an application fails to an unknown state, it is likely that an attacker may be able to exploit this indeterminate state to access unauthorized functionality, or worse create, modify or destroy data.

Fail safe

* Inspect the application’s fatal error handler.
* Does it fail safe? If so, how?
* Is the fatal error handler called frequently enough?
* What happens to in-flight transactions and ephemeral data?

Debug errors

* Does production code contain debug error handlers or messages?
* If the language is a scripting language without effective pre-processing or compilation, can the debug flag be turned on in the browser?
* Do the debug messages leak privacy related information, or information that may lead to further successful attack?

Exception handling

* Does the code use structured exception handlers (try {} catch {} etc) or function-based error handling?
* If the code uses function-based error handling, does it check every return value and handle the error appropriately?
* Would fuzz injection against the average interface fail?

Functional return values

Many languages indicate an error condition by return value. E.g.:

$query = mysql\_query(“SELECT \* FROM table WHERE id=4”, $conn);

if ( $query === false ) {

// error

}

* Are all functional errors checked? If not, what can go wrong?

Detailed error messages

Detailed error messages provide attackers with a mountain of useful information.

How to determine if you are vulnerable

* Are detailed error messages turned on?
* Do the detailed error messages leak information that may be used to stage a further attack, or leak privacy related information?
* Does the browser cache the error message?

How to protect yourself

Ensure that your application has a “safe mode” which it can return if something truly unexpected occurs. If all else fails, log the user out and close the browser window

Production code should not be capable of producing debug messages. If it does, debug mode should be triggered by editing a file or configuration option on the server. In particular, debug should not enabled by an option in the application itself

If the framework or language has a structured exception handler (ie try {} catch {}), it should be used in preference to functional error handling

If the application uses functional error handling, its use must be comprehensive and thorough

Detailed error messages, such as stack traces or leaking privacy related information, should never be presented to the user. Instead a generic error message should be used. This includes HTTP status response codes (ie 404 or 500 Internal Server error).

Logging

Where to log to?

Logs should be written so that the log file attributes are such that only new information can be written (older records cannot be rewritten or deleted). For added security, logs should also be written to a write once / read many device such as a CD-R.

Copies of log files should be made at regular intervals depending on volume and size (daily, weekly, monthly, etc.). .). A common naming convention should be adopted with regards to logs, making them easier to index. Verification that logging is still actively working is overlooked surprisingly often, and can be accomplished via a simple cron job!

Make sure data is not overwritten.

Log files should be copied and moved to permanent storage and incorporated into the organization's overall backup strategy.

Log files and media should be deleted and disposed of properly and incorporated into an organization's shredding or secure media disposal plan. Reports should be generated on a regular basis, including error reporting and anomaly detection trending.

Be sure to keep logs safe and confidential even when backed up.

Handling

Logs can be fed into real time intrusion detection and performance and system monitoring tools. All logging components should be synced with a timeserver so that all logging can be consolidated effectively without latency errors. This time server should be hardened and should not provide any other services to the network.

No manipulation, no deletion while analyzing.

General Debugging

Logs are useful in reconstructing events after a problem has occurred, security related or not. Event reconstruction can allow a security administrator to determine the full extent of an intruder's activities and expedite the recovery process.

Forensics evidence

Logs may in some cases be needed in legal proceedings to prove wrongdoing. In this case, the actual handling of the log data is crucial.

Attack detection

Logs are often the only record that suspicious behavior is taking place: Therefore logs can sometimes be fed real-time directly into intrusion detection systems.

Quality of service

Repetitive polls can be protocol led so that network outages or server shutdowns get protocolled and the behavior can either be analyzed later on or a responsible person can take immediate actions.

Proof of validity

Application developers sometimes write logs to prove to customers that their applications are behaving as expected.

Required by law or corporate policies

Logs can provide individual accountability in the web application system universe by tracking a user's actions.

It can be corporate policy or local law to be required to (as example) save header information of all application transactions. These logs must then be kept safe and confidential for six months before they can be deleted.

The points from above show all different motivations and result in different requirements and strategies. This means, that before we can implement a logging mechanism into an application or system, we have to know the requirements and their later usage. If we fail in doing so this can lead to unintentional results.

Failure to enable or design the proper event logging mechanisms in the web application may undermine an organization's ability to detect unauthorized access attempts, and the extent to which these attempts may or may not have succeeded. We will look into the most common attack methods, design and implementation errors as well as the mitigation strategies later on in this chapter.

There is another reason why the logging mechanism must be planned before implementation. In some countries, laws define what kind of personal information is allowed to be not only logged but also analyzed. For example, in Switzerland, companies are not allowed to log personal information of their employees (like what they do on the internet or what they write in their emails). So if a company wants to log a workers surfing habits, the corporation needs to inform her of their plans in advance.

This leads to the requirement of having anonymized logs or de-personalized logs with the ability to re-personalized them later on if need be. If an unauthorized person has access to (legally) personalized logs, the corporation is acting unlawful again. So there can be a few (not only) legal traps that must be kept in mind.

Logging types

Logs can contain different kinds of data. The selection of the data used is normally affected by the motivation leading to the logging. This section contains information about the different types of logging information and the reasons why we could want to log them.

In general, the logging features include appropriate debugging information’s such as time of event, initiating process or owner of process, and a detailed description of the event. The following are types of system events that can be logged in an application. It depends on the particular application or system and the needs to decide which of these will be used in the logs:

Reading of data file access and what kind of data is read. This not only allows to see if data was read but also by whom and when.

Writing of data logs also where and with what mode (append, replace) data was written. This can be used to see if data was overwritten or if a program is writing at all.

Modification of any data characteristics, including access control permissions or labels, location in database or file system, or data ownership. Administrators can detect if their configurations were changed.

Administrative functions and changes in configuration regardless of overlap (account management actions, viewing any user's data, enabling or disabling logging, etc.)

Miscellaneous debugging information that can be enabled or disabled on the fly.

All authorization attempts (include time) like success/failure, resource or function being authorized, and the user requesting authorization. We can detect password guessing with these logs. These kinds of logs can be fed into an Intrusion Detection system that will detect anomalies.

Deletion of any data (object). Sometimes applications are required to have some sort of versioning in which the deletion process can be cancelled.

Network communications (bind, connect, accept, etc.). With this information an Intrusion Detection system can detect port scanning and brute force attacks.

All authentication events (logging in, logging out, failed logins, etc.) that allow to detect brute force and guessing attacks too.

Noise

Intentionally invoking security errors to fill an error log with entries (noise) that hide the incriminating evidence of a successful intrusion. When the administrator or log parser application reviews the logs, there is every chance that they will summarize the volume of log entries as a denial of service attempt rather than identifying the 'needle in the haystack'.

How to protect yourself

This is difficult since applications usually offer an unimpeded route to functions capable of generating log events. If you can deploy an intelligent device or application component that can shun an attacker after repeated attempts, then that would be beneficial. Failing that, an error log audit tool that can reduce the bulk of the noise, based on repetition of events or originating from the same source for example. It is also useful if the log viewer can display the events in order of severity level, rather than just time based.

Cover Tracks

The top prize in logging mechanism attacks goes to the contender who can delete or manipulate log entries at a granular level, "as though the event never even happened!". Intrusion and deployment of rootkits allows an attacker to utilize specialized tools that may assist or automate the manipulation of known log files. In most cases, log files may only be manipulated by users with root / administrator privileges, or via approved log manipulation applications. As a general rule, logging mechanisms should aim to prevent manipulation at a granular level since an attacker can hide their tracks for a considerable length of time without being detected. Simple question; if you were being compromised by an attacker, would the intrusion be more obvious if your log file was abnormally large or small, or if it appeared like every other day's log?

How to protect yourself

Assign log files the highest security protection, providing reassurance that you always have an effective 'black box' recorder if things go wrong. This includes:

Applications should not run with Administrator, or root-level privileges. This is the main cause of log file manipulation success since super users typically have full file system access. Assume the worst case scenario and suppose your application is exploited. Would there be any other security layers in place to prevent the application's user privileges from manipulating the log file to cover tracks?

Ensuring that access privileges protecting the log files are restrictive, reducing the majority of operations against the log file to alter and read.

Ensuring that log files are assigned object names that are not obvious and stored in a safe location of the file system.

Writing log files using publicly or formally scrutinized techniques in an attempt to reduce the risk associated with reverse engineering or log file manipulation.

Writing log files to read-only media (where event log integrity is of critical importance).

Use of hashing technology to create digital fingerprints. The idea being that if an attacker does manipulate the log file, then the digital fingerprint will not match and an alert generated.

Use of host-based IDS technology where normal behavioral patterns can be 'set in stone'. Attempts by attackers to update the log file through anything but the normal approved flow would generate an exception and the intrusion can be detected and blocked. This is one security control that can safeguard against simplistic administrator attempts at modifications.

False Alarms

Taking cue from the classic 1966 film "How to Steal a Million", or similarly the fable of Aesop; "The Boy Who Cried Wolf", be wary of repeated false alarms, since this may represent an attacker's actions in trying to fool the security administrator into thinking that the technology is faulty and not to be trusted until it can be fixed.

How to protect yourself

Simply be aware of this type of attack, take every security violation seriously, always get to the bottom of the cause event log errors rather, and don't just dismiss errors unless you can be completely sure that you know it to be a technical problem.

Denial of Service

By repeatedly hitting an application with requests that cause log entries, multiply this by ten thousand, and the result is that you have a large log file and a possible headache for the security administrator. Where log files are configured with a fixed allocation size, then once full, all logging will stop and an attacker has effectively denied service to your logging mechanism. Worse still, if there is no maximum log file size, then an attacker has the ability to completely fill the hard drive partition and potentially deny service to the entire system. This is becoming more of a rarity though with the increasing size of today's hard disks.

How to protect yourself

The main defense against this type of attack are to increase the maximum log file size to a value that is unlikely to be reached, place the log file on a separate partition to that of the operating system or other critical applications and best of all, try to deploy some kind of system monitoring application that can set a threshold against your log file size and/or activity and issue an alert if an attack of this nature is underway.

Destruction

Following the same scenario as the Denial of Service above, if a log file is configured to cycle round overwriting old entries when full, then an attacker has the potential to do the evil deed and then set a log generation script into action in an attempt to eventually overwrite the incriminating log entries, thus destroying them.

If all else fails, then an attacker may simply choose to cover their tracks by purging all log file entries, assuming they have the privileges to perform such actions. This attack would most likely involve calling the log file management program and issuing the command to clear the log, or it may be easier to simply delete the object which is receiving log event updates (in most cases, this object will be locked by the application). This type of attack does make an intrusion obvious assuming that log files are being regularly monitored, and does have a tendency to cause panic as system administrators and managers realize they have nothing upon which to base an investigation on.

How to protect yourself

Following most of the techniques suggested above will provide good protection against this attack. Keep in mind two things:

Administrative users of the system should be well trained in log file management and review. 'Ad-hoc' clearing of log files is never advised and an archive should always be taken. Too many times a log file is cleared, perhaps to assist in a technical problem, erasing the history of events for possible future investigative purposes.

An empty security log does not necessarily mean that you should pick up the phone and fly the forensics team in. In some cases, security logging is not turned on by default and it is up to you to make sure that it is. Also, make sure it is logging at the right level of detail and benchmark the errors against an established baseline in order measure what is considered 'normal' activity.

Further Reading